Computers Programming
Course 12

Iulian Năstac

Recap from previous course

Strings in C

- The character string is one of the most widely used applications that involves vectors.
- A string in C is an array of char values terminated by a special null character value '\0'.

Recap from previous course

Standard functions for strings

- C provides a library for strings (**string.h**).
- C string library functions do not allocate space for the strings they manipulate, nor do they check that you pass in valid strings; it is up to your program to allocate space for the strings that the C string library will use.
- Calling ***string*** library functions with bad address values will cause a fault or "strange" memory access errors.

Recap from previous course

Some of the functions in the Standard C string library

- calculating the length of strings (eg strlen)
- copying the strings (eg strcpy)
- concatenating the strings (eg strcat)
- comparing the strings (eg strcmp)
- searching (identification) of character or substrings (eg strchr, strstr)

Recap from previous course

1. Computing the string length

- The length of a string is the number of the own characters which fall within its composition.

- NULL character is not considered in determining the length of a string.

Syntax:

unsigned strlen(const char * s);

Recap from previous course

2. Copying the strings

- Sometimes it is necessary to copy a string of character from a memory area in another part. For this we use the strcpy function, which has the syntax:

char * strcpy (char * dest, const char* source);

- The function makes a copy of the string pointed by the **source** to the memory area whose address is indicated by **dest**.
- The function copies all the characters, including the null one, from the end of the string.
- The function returns the address of **dest**.

```
#include<stdio.h>
#include<string.h>
#define MAX 100      /* it is assumed that a word has no more than
100 characters*/
int main( )
{
 int max=0, i;
 char word[MAX+1];
 char word _max[MAX+1];
 while (scanf("%100s", word)!=EOF)
          if (max < (i = strlen(word)))
                    { max= i;
                       strcpy (word _max, word);
                    }
 if (max)  printf("The longest word is %s and its length is %d \n",
                              word _max, max);
getch();
}
```

Recap from previous course

3. Concatenating the strings

- For concatenating strings we can use **strcat** function.

- Syntax:

char *strcat(char *dest, const char *source);

Variant of **strcat** function

char *strncat (char *dest, const char *source, unsigned n);

- If *n* $\geq$ *length of source* - then all characters of the source string are concatenated after **dest**;

- If *n* < *length of source* - then only the first *n* characters of the source string are concatenated after **dest**.

Recap from previous course

4. Comparing the strings

- Strings of characters can be compared using the ASCII characters in their structure.

- To better understand this mechanism we consider two strings: **s**1 and **s**2.

- We have the following cases:

two strings: **s**1 and **s**2

- **s1 = s2** – if both have similar length and s1[i] = s2[i] ∀ i.

- **s1 < s2** – if ∃ ***i*** such that *s1[i]<s2[i]* and *s1[j]=s2[j]* ∀ j= 0, 1, …, i-1.

- **s1 > s2** – if ∃ ***i*** such that *s1[i]>s2[i]* and *s1[j]=s2[j]* ∀ j= 0, 1, …, i-1.

Recap from previous course

strcmp function

int strcmp(const char *s1, const char *s2);

- This function returns:
  - a negative value - if s1 < s2
  - 0 - if s1 = s2
  - a positive value - if s1 > s2

```
#include<stdio.h>
#include<string.h>
#define MAX 100

int main ()
{ char word[MAX+1];
  char word _max[MAX+1];
 word _max [0]='\0'; /* word _max is initialized with null */

  while (scanf("%100s", word)!=EOF)
        { if (stricmp(word, word _max)>0)
              strcpy (word _max, word);
         }
   printf("\n The greatest word is  %s", word _max);

   getch();
   return 0;
}
```

Recap from previous course

5. Finding strings or characters

- Some useful functions:

strchr(s1, ch); /* return a pointer at first finding of ***ch*** character in ***s1*** */

strstr(s1, s2); /* return a pointer at first finding of ***s2*** inside ***s1*** */

- If there is no match (finding), then these functions return zero.

Recap from previous course

Syntax:

char *strchr(const char *s1, const char ch);

char *strstr(const char *s1, const char *s2);

Example

```
#include<stdio.h>
#include<string.h>
...

int main ()
{
 char s[80];
 gets(s);          /* write a text */
...
 if (strchr(s, 'e')) printf("The character - e - was find in the string");

 if (strstr(s, "test")) printf("The word - test -  was find in the string");
...
}
```

Recap from previous course

Initializing Vectors

- In C, a vector can be initialized, either one by one element, or using a single statement as follows:

```
double A[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

- The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ].

Example

- Computing the sum of a vector elements

```
#include<stdio.h>
#include<conio.h>

int main()
{
 int Vect[100], n, i, S=0;
 do
    {
     printf("\n\n Establish the number of elements \n\t n = ");
     scanf("%d", &n);
    }while((n<1) || (n>100));

printf("\n\n  Introduce the elements");
 for(i=0; i<n; i++)
       {
        printf("\n Vect[%d] = ",i+1);
        scanf("%d", &Vect[i]);
       }
```

```
 printf("\n\nComputing the summ");

 for(i=0; i<n; i++)
        {
          S=S+Vect[i];
        }
 printf("\n S = %d", S);

 getch();
 return 0;
}
```

Two-Dimensional Arrays

- The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size **lim_1**, **lim_2** you would write something as follows:

type *arrayName*[lim_1][lim_2]

How to access the matrix elements

- By indices
- By pointers

Example

- **int a[3][4];**

  this declares an integer array of 3 rows and 4 columns. Index of row will start from 0 and will go up to 2.

	Column 0	Column 1	Column 2	Column 3
row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Example

- Load all numbers from 1 to 12 in a two-dimensional array and then display this matrix, line by line.

```
#include <stdio.h>
#include <conio.h>

int main()
{  int t,i,num[3][4];
   for(t=0;t<3;t++)
            for(i=0;i<4;i++)
                        num[t][i]=(t*4)+i+1;
/* then display */
   for(t=0;t<3;t++)
            { for(i=0;i<4;i++) printf("%3d\t",num[t][i]);
              printf("\n");
            }

 getch();
 return 0;
}  /* STOP */
```

rows / columns	*0*	*1*	*2*	*3*
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Notes:

- Two-dimensional arrays are stored in row-column form, wherein the first index (at the left) shows the line, and the second (on the right) specifies the column.
- The index on the right changes faster than the left one, when counting matrix elements in the order they are actually stored in memory.
- The same goes for multidimensional arrays (with more than two dimensions).

The total size in bytes:

No_of_bytes = sizeof(type) * lim_1*lim_2

Two-dimensional matrices as function parameters

- When a two-dimensional matrix is used as function parameter, then a pointer to the matrix is transmitted.

- Furthermore, a two-dimensional array parameter must define at least the number of columns (lim_2), since the C compiler needs to know the length of each row in order to properly index the matrix.

A function that receives a two dimensional array of integers with size 10 x 10 is declared:

```
void funct(int x[ ][10])
        {
        ...
        }
```

- *It can also specify the size of the left, but not necessarily.*
- *However, the compiler must know the size of the right part in order to properly execute expressions like x[2][4] inside the function.*
- *If the row size is not known, the compiler cannot determine where is the start of the second line (the third, the fourth, etc.).*

Example

- Write a program to store notes for each student from three different groups. Suppose we have a maximum of 20 students in each group. We will use a two-dimensional array as a database.

```
# include<stdio.h>
# include<conio.h>
# include<stdlib.h>

# define GROUPS 3
# define STUDENTS 20

int note[GROUPS][STUDENTS];

void introduce_grades(void);
int take_grades(int num);
void display_grades(int g[ ][STUDENTS]);

int main()
{
 char ch;
```

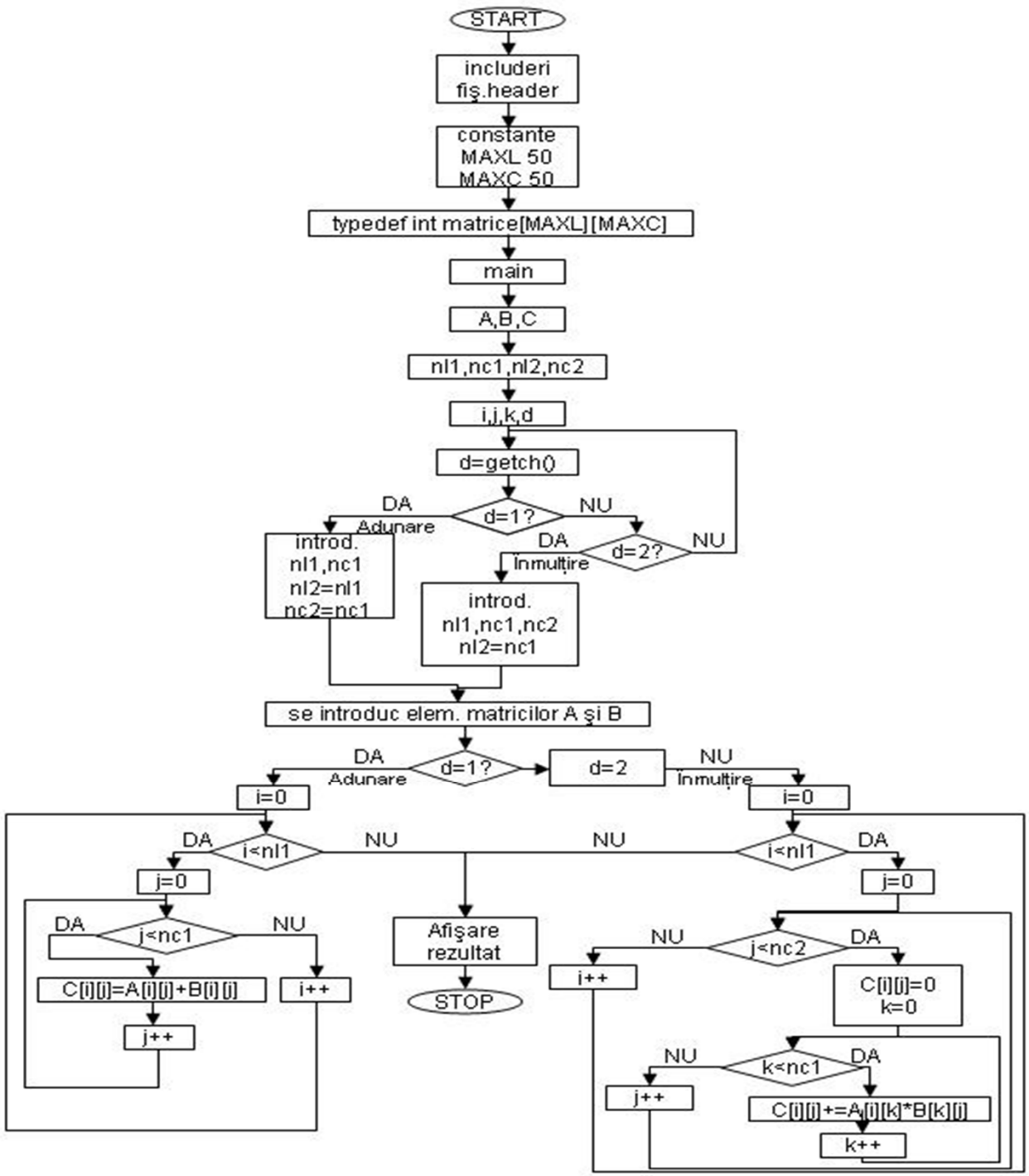
```
 for( ; ; )
 {
   do
    {
     printf("\n\n(I)ntroduce grades\n");
     printf("(P)resent grades\n");
     printf("(E)xit\n");
     ch=getch( );
    } while((ch != 'I') && (ch != 'P') && (ch != 'E'));

 switch(ch)
            {
            case 'I': introduce_grades(); break;
            case 'P':display_grades(note); break;
            case 'E':printf("\n\n\n\n\t\t\t\tSTOP PROGRAM");
                      printf("\n\n\n\n\t\t\t\t press a key");
                     getch();
                     exit(0);
            }
 }
 return 0;
}
```

```
void introduce_grades(void)
{
int t, i;
for(t=0; t<GROUPS; t++)
        {
        printf("\nGroup # %d: \n",t+1);
        for(i=0; i<STUDENTS; i++)
                note[t][i]=take_grades(i);
        }
}


int take_grades(int num)
{
   char s[80];
   printf("Enter grade for student %d: ",num+1);
   gets(s);
   return(atoi(s));
}
```

```
void display_grades(int g[ ][STUDENTS])
{
int t,i,s;

for(t=0; t<GROUPS; t++)
  {
   printf("\n\nGroup #%d: \n",t+1);
   for(i=0; i<STUDENTS; i++)
              {
             printf("Student %d has %d\n",i+1,g[t][i]);
             s=(i+1)%10;
             if(s == 0)
                        {
                        printf("\t\t\t Press a key!\n");
                        getch();
                        }
             }
  }
printf("\n\n\tPresentation of grades is completed. Press a key!");
getch();
}
```

Example

- Program for addition or multiplication of two matrices


START
includeri
fiş.header
constante
MAXL 50
MAXC 50
typedef int matrice[MAXL][MAXC]
main
A,B,C
nl1,nc1,nl2,nc2
i,j,k,d
d=getch()
DA
d=1?
NU
Adunare
introd.
nl1,nc1
nl2=nl1
nc2=nc1
DA
d=2?
NU
Înmulţire
introd.
nl1,nc1,nc2
nl2=nc1
se introduc elem. matricilor A şi B
DA
d=1?
d=2
NU
Adunare
Înmulţire
i=0
i=0
DA
i<nl1
NU
NU
i<nl1
DA
j=0
j=0
DA
j<nc1
NU
C[i][j]=A[i][j]+B[i][j]
j++
i++
Afişare
rezultat
STOP
NU
j<nc2
DA
i++
C[i][j]=0
k=0
NU
k<nc1
DA
j++
C[i][j]+=A[i][k]*B[k][j]
k++

```
#include<stdio.h>
#include<conio.h>
#define MAXL  50
#define MAXC  50

typedef int  matrice [MAXL][MAXC];

int main()
{
  matrice A,B,C;
  int nl1,nc1;
  int nl2,nc2;
  int i,j,k;
  char d;
  clrscr();
  printf("The program allows the addition or multiplication of two
matrices.\n"
            "1 - Addition\n2 - Multiplication\n");
  d=getch();
  while((d!='1') && (d!='2'))  d=getch();
  clrscr();
```

```
printf("Enter the numbers of rows and columns (0,50)\n");
 switch(d)
  {
    case'1':
              printf("number of rows :");
              scanf("%d",&nl1); nl2=nl1;
              printf("numbers of columns :");
              scanf("%d",&nc1); nc2=nc1;
              break;
    case'2':
              printf("numbers of rows for matrix A:");
              scanf("%d",&nl1);
              printf("numbers of columns for matrix A:");
              scanf("%d",&nc1); nl2=nc1;
              printf("numbers of columns for matrix B:");
              scanf("%d",&nc2);
              break;
  }
 clrscr();
```

```
printf("Enter the matrix elements.");
 for(i=0;i<nl1;i++)
    for(j=0;j<nc1;j++)
     {
            printf("A(%d,%d)=",i+1,j+1);
            scanf("%d",&A[i][j]);
     }
 for(i=0;i<nl2;i++)
    for(j=0;j<nc2;j++)
     {
            printf("B(%d,%d)=",i+1,j+1);
            scanf("%d",&B[i][j]);
     }
```

```
switch(d)
   {
     case'1':
              for(i=0;i<nl1;i++)
                    for(j=0;j<nc1;j++)
                      C[i][j]=A[i][j]+B[i][j];
              break;

     case'2':
             for(i=0;i<nl1;i++)
               for(j=0;j<nc2;j++)
                     {
                       C[i][j]=0;
                       for(k=0;k<nc1;k++)
                           C[i][j]+=A[i][k]*B[k][j];

                     }
              break;
   }
```

```
 printf("The resulting matrix is:\n");
  for(i=0;i<nl1;i++)
   {
    for(j=0;j<nc2;j++)
       printf("%5d  ",C[i][j]);
  printf("\n"); /* putch('\n');*/
   }
  putch('\n');

  getch();
  return 0;
}
```

Matrices of strings

- The use of arrays of strings is a process commonly used in programming.
- For example, a specific process in a database is to compare the commands given by the user with a matrix of commands.
- To create an array of strings, one can simply use a two-dimensional array of character.

Notes

- Left Index - refers to the number of rows
- Right Index - refers to the maximum dimension of each string (including NULL character)

Example

char MAT[30][80];

- It is very easy to gain access to an individual row in the matrix. You have to specify only the left index. For example, the following call gets access to the third row of matrix_string:

gets(MAT[2]);

which is functionally equivalent to:

gets(&MAT[2][0]);

Example

- Write a simple text editor

```
# include<stdio.h>
# include<conio.h>
# include<stdlib.h>

#define MAX 100
#define LUNG 80
char text[MAX][LUNG];

int main()
{
  register int i,j,t;
  printf("\n Enter a blank line to exit the editor.\n");
  for(t=0;t<MAX;t++)
          { printf("%d: ",t);
            gets(text[t]); /*for each line the characters are inserted till pushing
                             ENTER */
                           /* equivalent with gets(&text[t][0]); */
            if (!*text[t]) break;/*exits the program if the string contains only null /0*/
          }
```

```
 putchar('\n');
  for(i=0;i<t;i++) /* redisplay previously entered text */
          { for(j=0;text[i][j];j++) putchar(text[i][j]);
            putchar('\n');
          }

  getch();
  return 0;
}
```

Multidimensional arrays

- The C language allows the use of arrays with more than two dimensions. Maximum size could depend, sometimes, by the version of the compiler.
- The general format of a multidimensional array is:

type name[lim_1][lim_2]...[lim_n];

- However, arrays of more than two sizes are not used very often due to the large amount of memory required.

Initializing Arrays

- In C, an array can be initialized, either one by one element, or using a single statement as follows:

type MAT[lim_1][lim_2]...[lim_n]={list};

- The number of values between braces { } can not be larger than the sum of the number of elements that we declare for the array between square brackets [ ].

```
int MAT [3][2]  =  {   1, 1,
                       2, 4,
                       3, 5
                   };
```